Three Pillars improving Vision Foundation Model Distillation for Lidar

Supplementary Material

Gilles Puy[1] Spyros Gidaris[1] Alexandre Boulch[1] Oriane Siméoni[1]
Corentin Sautier[1,3] Patrick Pérez[2] Andrei Bursuc[1] Renaud Marlet[1,3]

[1]valeo.ai, Paris, France [2]Kyutai, Paris, France
[3]LIGM, Ecole des Ponts, Univ Gustave Eiffel, CNRS, Marne-la-Vallée, France

In this supplementary material, we provide detailed robustness results in Appendix A, give training details in Appendix B, visually inspect the properties of the ScaLR feature space in Appendix C, present preliminary results for object detection in Appendix D, and discuss some limitations of our work in Appendix E.

A. Robustness

We present in Tab. 9 a detailed version of Tab. 6 with the mIoU% attained for the eight different types of corruptions considered in [31].

We observe that WI-768 pretrained on multiple datasets is second or third for each type of corruption, except for motion blur, which permits it to achieve the best overall mCE% and mRR%.

B. Training Details

B.1. PandaSet

PandaSet [69] is made of scans acquired in San Francisco and along El Camino Real from Palo Alto to San Mateo. We use the scans collected in San Francisco as train set and the rest as validation set. We also separate the scans collected with the Pandar64 lidar from the scans collected with the PandarGT and treat them as different datasets. Finally, we merge the fine-grained original classes into 17 classes similar to those used in nuScenes and SemanticKITTI. These classes are: road, traffic sign, barrier, pedestrian, vegetation, road marking, sidewalk, manmade, traffic cone, car, motorcycle, truck, bus, bicycle, other vehicle, ground, and driveway.

B.2. Pretraining

Shared setting. The point tokens (see [51] for details) in the WaffleIron backbones are computed using 16 nearest neighbors and the following point features: lidar intensity, 3D Cartesian xyz coordinates, radius/range. The field-of-view in the spatial mixing blocks is restricted to $[-64\text{ m}, +64\text{ m}]$ along the x and y axes and $[-8\text{ m}, +8\text{ m}]$ along the z axis; we use a grid of resolution 50 cm.

We pretrain the WaffleIron backbones using AdamW [40] with a weight decay of 3×10^{-4} and a learning rate linearly increasing from 0 to 0.002, then decreasing to 10^{-5} following a cosine schedule. The number of iterations or epochs corresponding to that maximum learning rate, as well as the total number of iterations or epochs, are described below, depending on the setting.

Mono-dataset setting. The mono-dataset setting is the setting used in Secs. 3.3, 4.3, 4.4, 4.7. We pretrain the WaffleIron backbones by distilling 2D features during 19 epochs, with a batch size of 16. The learning rate reaches its maximum value after 2 epochs. The MinkUNet backbone is pretrained following SLidR [58] protocol with the following adjustments: we use a batch size of 12, an initial learning rate of 1.5 — the optimizer used in SLidR is SGD — and 25 epochs.

Multi-dataset setting. The multi-dataset setting is the setting used in Secs. 4.5, 4.6. The pretraining dataset contains: 28,130 scans for nuScenes, 19,130 scans for SemanticKITTI, 3,920 scans for PandaSet-64 and 3,920 scans for PandaSet-GT. We pretrain WI-256 by distilling 2D features with a batch size of 16 during 19 epochs on nuScenes, 28 epochs on SemanticKITTI, 136 epochs on PandaSet-64 or PandaSet-GT, 11 epochs on nuScenes & SemanticKITTI, 10 epochs on the mix of all datasets. The number of epochs is adjusted so that the backbone is pretrained for approximately the same number of iterations. For WI-768, as the available GPU memory is not sufficient for some batches, we decrease the batch size to 8 and pretrain for 49 epochs on nuScenes and 25 epochs on the mix of all datasets. The learning rate reaches its maximum value after 3500 iterations in all cases.

2D teacher. All the results presented in this paper with the DINO-pretrained ViT-S/8 are obtained by distilling the keys at the last attention layer. This choice was guided by the fact that the last keys have properties that enable the design of unsupervised object discovery algorithms [62, 63, 66]. The results obtained with the DINOv2-pretrained ViTs are obtained by distilling the features before the last normalization layer.

Method	2D Back.	3D Back.	Pretrain. dataset	mCE% ↓	mRR% ↑	Corruptions (mIoU% ↑)							
						Fog	Wet	Snow	Motion	Beam	Cross	Echo	Sensor
–	–	Cylinder3D [84]	–	*105.6*	*78.1*	*61.4*	*71.0*	*58.4*	*56.0*	*64.2*	*45.4*	*60.0*	*43.0*
–	–	2DPASS [71]	–	*98.6*	*75.2*	*64.5*	*76.8*	*54.5*	*62.0*	*67.8*	*34.4*	*63.2*	*45.8*
–	–	SPVCNN [64]	–	97.5	*75.1*	*55.9*	*74.0*	*42.0*	*74.6*	*69.0*	*28.1*	*65.0*	*51.6*
–	–	GFNet [52]	–	*92.6*	*83.3*	*69.6*	*75.5*	*71.8*	*59.4*	*64.5*	*66.8*	*61.9*	*42.3*
–	–	WI-768	–	90.9	80.6	72.2	78.0	66.6	55.2	70.4	48.7	64.7	52.4
PPKT [38]	ResNet-50	MinkUNet	nuScenes	*105.6*	*76.1*	*64.0*	*72.2*	*59.1*	*57.2*	*63.9*	*36.3*	*60.6*	*39.6*
SLidR [58]	ResNet-50	MinkUNet	nuScenes	*106.1*	*76.0*	*65.4*	*72.3*	*56.0*	*56.1*	*62.9*	*41.9*	*61.2*	*38.9*
Seal [39]	ResNet-50	MinkUNet	nuScenes	*92.6*	*83.1*	*72.7*	*74.3*	*66.2*	*66.1*	*66.0*	*57.4*	*59.9*	*39.9*
ScaLR (ours)	ViT-L/14	WI-768	nuScenes	89.1	83.7	70.8	77.2	67.1	55.9	70.0	65.7	63.9	51.1
ScaLR (ours)	ViT-L/14	WI-768	Multiple	87.4	83.8	72.2	77.9	69.1	57.4	70.1	62.7	64.0	52.2

Table 9. **Robustness to corruptions**. The evaluation is conducted on nuScenes-C from the Robo3D benchmark [31]. We report the mCE%, mRR%, and the mIoU% attained for the eight corruptions, i.e., fog, wet ground, snow, motion blur, beam missing, crosstalk (among multiple sensors), incomplete echo, and cross-sensor (beam and point dropping). The scores in italic are obtained from [31, 39]. PPKT, SLidR and Seal use a MoCov2 ResNet-50. We use DINOv2 ViT-L/14.

Multi-teacher distillation. Let us denote the output feature dimension of both image teachers by $F_{2D}^{(1)}$ and $F_{2D}^{(2)}$, respectively. On the image side, we ℓ_2-normalize the pixel features extracted by each teacher and concatenate them. On the point cloud side, the head ψ_{3D} is a 2-layer MLP where the hidden linear layer has size $2 \times F_{3D}$ and is followed by a Layer Norm and a `ReLU`. The final linear layer of the MLP has size $F_{2D}^{(1)} + F_{2D}^{(2)}$ to match the size of the concatenated 2D features. These point features are then split into two parts of size $F_{2D}^{(1)}$ and $F_{2D}^{(2)}$, respectively. Each part is ℓ_2-normalized independently. The normalized features are then re-concatenated. Finally, we distill the knowledge of the 2D features by applying Eq. (1) directly on the features of size $F_{2D}^{(1)} + F_{2D}^{(2)}$.

B.3. Linear probing

The linear head is trained with a batch size of 8, using AdamW with a weight decay of 3×10^{-3}. The learning rate linearly increases from 0 to 0.001 during the first 2 epochs and then decreases to 10^{-5} following a cosine schedule. We use 20 epochs on nuScenes and SemanticKITTI, and 50 epochs on PandaSet-64 and PandaSet-GT.

B.4. Finetuning

For finetuning the pretrained WaffleIron backbones, we use a batch size of 8, using AdamW without weight decay. The learning rate linearly increases from 0 to 0.002 during the first tenth of epochs and then decreases to 0 following a cosine schedule. During finetuning, we also use stochastic depth [27] with a layer drop probability of 0.2. We finetune the WaffleIron backbones for 45 epochs and a layer-wise learning rate decay parameter of 0.95 when using 1% and 10% of available data, and for 25 epochs and a layer decay parameter of 0.99 when all annotated data are available.

C. Visual Inspection

We provide a visualization of the features computed by a ScaLR-pretrained ϕ_{3D} backbone in Fig. 2. We use our WI-768 pretrained on nuScenes, SemanticKITTI, PandaSet-64 and PandaSet-GT. In this figure, the features are projected onto the space spanned by their 3 principal components and used as RGB values to color the point clouds. Note that the PCA is done independently on each scan, which explains why the colors are not consistent from one scan to another.

We notice that the feature space of our pretrained backbone is correctly structured as we can distinguish rather easily the main urban constructions and objects in these figures. For example, we notice that the points belonging to road and sidewalk have similar colors (per scan) on nuScenes and SemanticKITTI. On PandaSet-64 and PandaSet-GT, we also notice that the cars have similar colors (per scan) as well. Let us mention that the road on PandaSet-GT scans appears in a less uniform color than on the other datasets. It could be explained by a higher density of points on the road for this lidar, which might lead to more subtle differences between features after distillation and/or PCA.

We continue our visual inspection of the distilled features by presenting feature similarity map with respect to class prototypes in Fig. 3. The features are extracted at the output of ϕ_{3D} and are ℓ_2-normalized. The similarity maps are then obtained as follows. For each scan, we use the ground-truth labels to extract the point features of a class of interest (car, pedestrian, road or sidewalk). We average all the corresponding features to obtain a single class prototype for that class. Finally, we compute the similarity of all point features with respect to this class prototype. This is a similar procedure as the one used in Fig. 1 but using a mean feature instead of a single point feature.

In all cases, we notice that the most similar features to a class prototype belongs to the corresponding class, as expected. This is another indication that the feature space is well structured where: the features of a same semantic class are close to each other; the features of two different semantic class are well separated. Nevertheless, when inspecting closely the similarity map, we notice sometimes some "leakage" around the objects of interest. This phenomenon is mostly visible for the class pedestrian. We believe that these artifacts are due to errors when projecting the points onto the camera plane, which affects the boundary of the objects. Finally, we remark as well that the similarity maps are less sharp on PandaSet-64 and PandaSet-GT than on nuScenes and SemanticKITTI, likely because of the small number of scans available in PandaSet.

D. Preliminary Results on Object Detection

Our results so far have shown the quality of our ScaLR features for semantic segmentation. In this section, we evaluate if these features can be useful for another task: object detection.

As WaffleIron is a backbone originally designed for semantic segmentation [51], we adapted it to object detection by modifying PointRCNN [60] to accept WaffleIron instead of PointNet++. Note that a similar approach was used in [58] but with MinkUNet instead of WaffleIron. We then trained this new object detection backbone on KITTI detection [20] using the WI-256 backbone pretrained on all four considered datasets, with ScaLR or with a non-pretrained WI-256. This experiment was conducted using the OpenPCDet toolbox [65] using the default configuration file of PointRCNN but using a batch size of 2 (with a maximum learning rate of 0.001) to allow training on one NVIDIA GeForce RTX 2080 Ti.

The results in Tab. 10 show that our ScaLR pretraining method significantly improves object detection results too. Let us nevertheless mention that our object detection backbone is non-standard (modified PointRCNN) and that some modifications of ScaLR will be necessary if one wants to pretrain better-performing backbones such as, e.g., PV-RCNN [61], for which it is common practice to pretrain the backbone after the BEV projection modules (see, e.g., [6]), where we loose direct mapping between point and pixel features, making Eq. (1) not directly applicable.

E. Limitations

Our study in Sec. 3.3 shows that the linear probing mIoU has a standard deviation around 1.0 percentage point between different pretrainings. Some possibilities to reduce these small fluctuations might be to explore longer pretraining schedules, or re-increase the number of loaded images per scan (from 1 to 6).

AP@R40 (%)	Car	Pedestrian	Cyclist
No pretrain.	71.1	49.0	49.1
ScaLR (ours)	**83.0**	**58.0**	**71.8**

Table 10. **Object detection.** Performance of our modified PointRCNN on KITTI detection [20]. We compare the performance reached by a non-pretrained backbone and a backbone pretrained using ScaLR.

Beides, our work raises the possibility of replicating undesirable biases present in the large pretrained 2D models used for distillation. These models are known to harbor problematic biases related, e.g., to geographic location, gender, skin tone, and age. When distilling these 2D vision models into 3D lidar models, there is a potential for these biases to be amplified or mirrored. Our resulting lidar models may exhibit varying performance across different geographical regions, influenced by how these regions are represented in the training datasets of the original 2D models and in the 2D-to-3D distillation training data. For real-world applications of this distillation strategy, practitioners are expected to be mindful about the 2D foundation model used and the nature of the data it was trained upon (e.g., potential biases, privacy breaches, licenses, etc.)

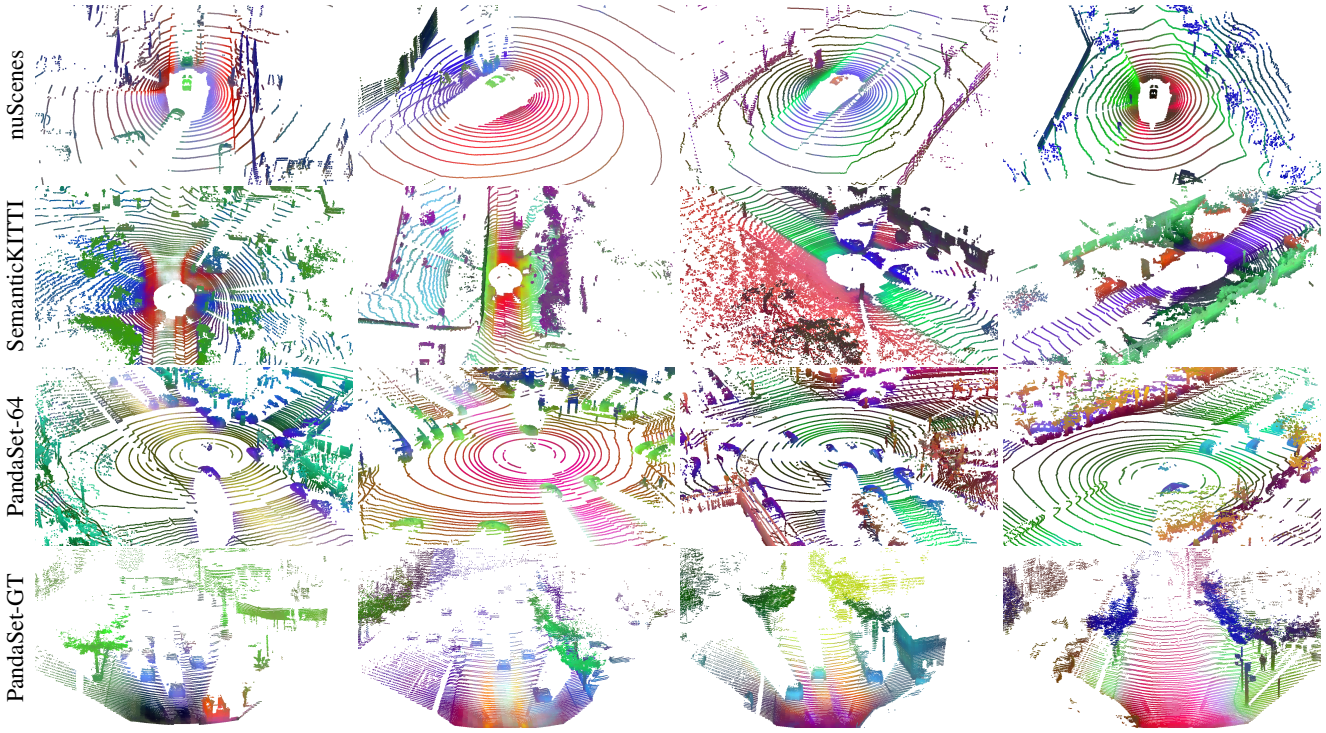

Figure 2. **Distilled feature visualizations**. We project the features at the output of ϕ_{3D} into a three-dimensional space by PCA. The projected value serves as RGB value to color the point clouds, i.e., the first, second and third components are used as the red, green and blue channels, respectively. Note that the PCA is done independently for each scan, which explains why the colors are not consistent from one scan to another. In this figure, we used the WI-768 pretrained on nuScenes, SemanticKITTI, PandaSet-64 and PandaSet-GT with ScaLR.

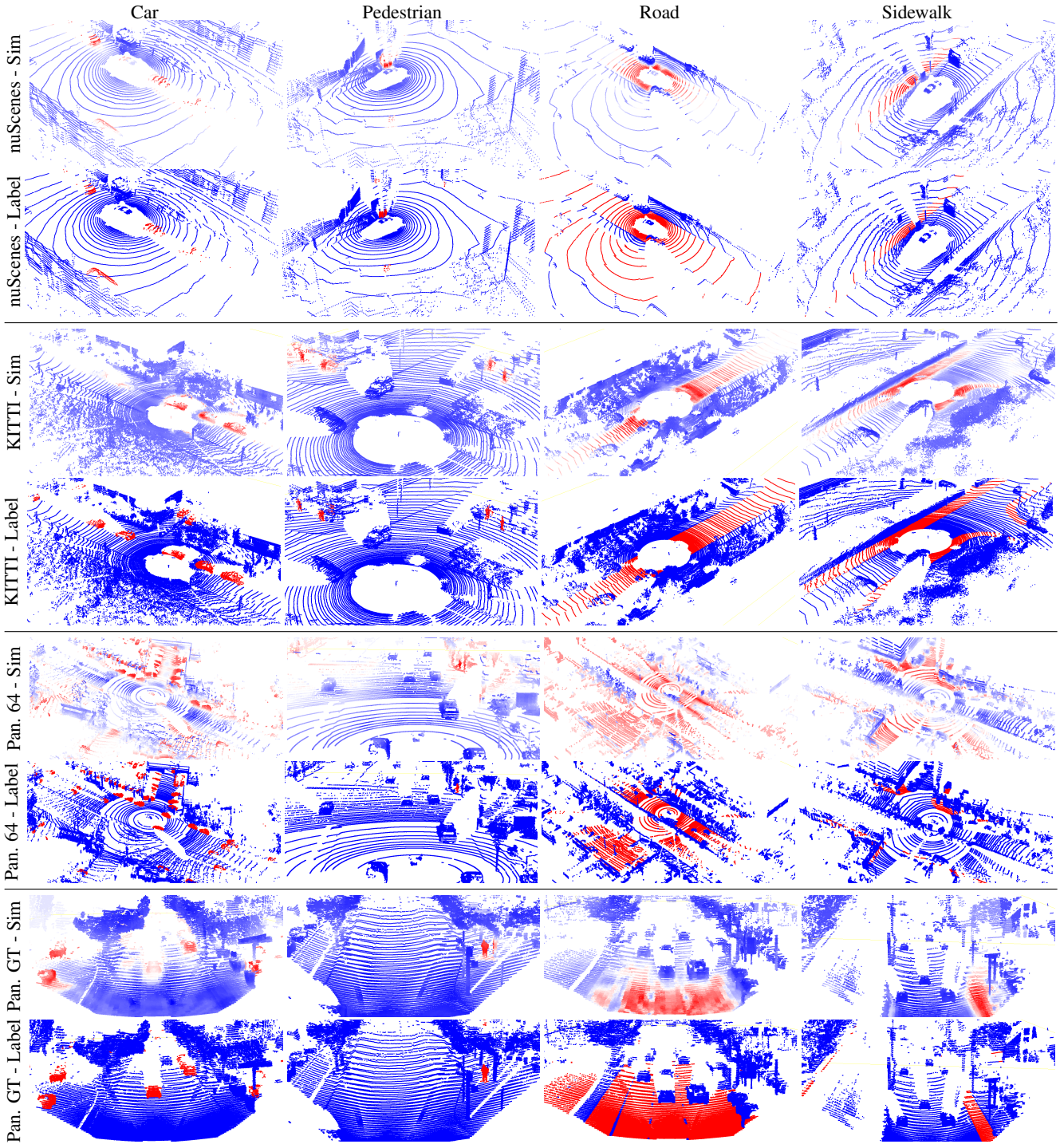

Figure 3. **Similarity map with class prototype**. For each scan, we use the ground-truth labels (presented on even rows) of four classes (car, pedestrian, road, sidewalk) to compute a class prototype (mean feature of the point belonging to the considered class). We then compute the feature similarity map (presented on odd rows) with respect to that class prototype. Color goes from blue to red for low and high values.